Von der Programmiersprache zu den Nullen und Einsen

Dass der Code unserer Dosenampel durch einen **Binärcode** (also einen Code mit zwei verschiedenen Zeichen; in unserem Falle die Zeichen 0 und 1) dargestellt werden kann, haben wir in den letzten Stunden gesehen. Heute sehen wir nun endlich, dass auch ganz „normale“ Programmiersprachen in logischen Schritten in Nullen und Einsen umgewandelt werden können. Wir verfolgen dabei in einfachen Schritten den **Weg von der Programmiersprache hin zu den Nullen und Einsen**, mit denen Computer oder Smartphones etc. heutzutage im Kern – immer noch – arbeiten.

Höhere Programmiersprachen

```
int main() {
    int a = 2;
    int b = 3;
    int c = a + b;
    return c;
}
```

Wenn man heute von „ProgrammiererInnen“ spricht, meint man Leuten, die **Software** erstellen (Betriebssysteme, Spiele, PC-Programme, …). Dazu benutzen sie **höhere Programmiersprachen**. Es ist nicht unüblich, dass ProgrammiererInnen mehrere Sprachen beherrschen (z. B. C++, Java, …). **Programmiersprachen** sind für uns Menschen **gut lesbar**, da sie meist **englische Wörter** beinhalten, die dann den **Programmcode** ergeben. Die Programme, in denen man programmiert, nennt man auch **„integrierte Entwicklungsumgebung“**. Sie **können den Programmcode unter anderem direkt in funktionierende Software umwandeln**. Von diesem Prozess merkt man nichts.

⬇ … kann aber auch - rein aus Neugier - umgewandelt oder direkt geschrieben werden in Assemblersprachen …

Von Assemblersprachen zu Binärcode

zugehöriger Assemblercode

```
push rbp
mov rbp, rsp
mov DWORD PTR [rbp-4], 2
mov DWORD PTR [rbp-8], 3
mov eax, DWORD PTR [rbp-8]
mov edx, DWORD PTR [rbp-4]
add eax, edx
mov DWORD PTR [rbp-12], eax
mov eax, DWORD PTR [rbp-12]
pop rbp
ret
```

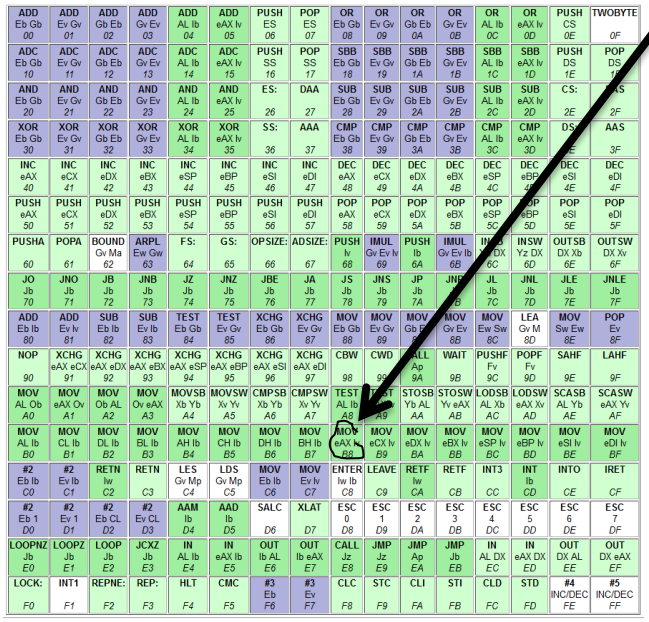
Eine **Assemblersprache** ist für uns Menschen zwar auch noch **gut lesbar**, ist aber schon sehr an den verbauten **Prozessor im PC gebunden**. Sie ist deshalb eine **maschinenorientierte Sprache**. Im Prozessor werden Nullen und Einsen „verrechnet“. Da es jedoch zig verschiedene Prozessoren (Intel, AMD, …) und noch mehr Modelle gibt, **sieht die Assemblersprache immer anders aus**. Die Programme, in denen man in Assemblersprache schreibt, heißen auch **Assembler**. Sie **können den Assemblercode direkt in funktionierende Software umwandeln**. Von diesem Prozess merkt man nichts. Assemblersprachen sind älter als die höheren Programmiersprachen, mit denen man heutzutage programmiert. Aber sie helfen uns dabei, den Weg zu den Nullen und Einsen besser zu verstehen!

Zum Verständnis schauen wir uns das Beispiel links also genauer an!

Assemblersprachen benutzen in ihrem Code **mnemonische Symbole** wie z. B. „mov“, die dem Prozessor letztlich sagen, was mit den Nullen und Einsen passieren soll. Unser Befehl wird wohl so etwas tun wie die Nullen und Einsen im Prozessor an einen anderen Ort zu kopieren (englisch „to move“).Diese mnemonischen Symbole können wir uns **gut einprägen** (das Wort „mnemo“ kann auch mit „Merkhilfe“ übersetzt werden).

Auf der Rückseite geht es weiter!

Intel x86 Assembler Instruction Set Opcode Table



Alle mnemonischen Symbole sind wiederum durch Zahlen - so genannte Opcodes - durchnummeriert. **Unser Befehl „mov“ hat bei diesem Prozessor die Zahl bzw. den Opcode „B8“**, weil es der Hersteller des Prozessors so festgelegt hat. Auf dem Weg zur funktionierenden Software wandelt der Assembler nun alle mnemonischen Symbole im Programmcode automatisch in Opcodes - also Zahlen - um. Neben den mnemonischen Symbolen werden auch die anderen Bestandteile des Assemblercodes umgewandelt.

Doch warum ist „B8“ eine Zahl? Ganz einfach! Diese **hexadezimale Zahl** kann ganz einfach auch in Nullen und Einsen umgewandelt bzw. dargestellt werden. **Die hexadezimalen Opcodes sind also ein wichtiger „Schlüssel“ zu den Nullen und Einsen.**

Dez	Hex	Okt	Binär
176	B0	260	10110000
177	B1	261	10110001
178	B2	262	10110010
179	B3	263	10110011
180	B4	264	10110100
181	B5	265	10110101
182	B6	266	10110110
183	B7	267	10110111
184	B8	270	10111000
185	B9	271	10111001
186	BA	272	10111010
187	BB	273	10111011
188	BC	274	10111100
189	BD	275	10111101
190	BE	276	10111110
191	BF	277	10111111

Anhand einer Tabelle wandeln wir nun unseren hexadezimalen Opcode ganz einfach in Nullen und Einsen um. „B8“ ist also „10111000“.

Schön und gut. Wir können unseren Programmcode in Nullen und Einsen umwandeln. Was passiert nun?

Einzelne Opcodes (oder Opcodes mit zusätzlichen Angaben wie z. B. Adressen) in binärer Darstellung stellen direkt **Maschinenbefehle** dar. Maschinenbefehle sagen dem Prozessor, wie er Nullen und Einsen verarbeiten soll, so dass wir - über weitere Zwischenschritte - Informationen auf dem Bildschirm sehen oder uns z. B. Musik am Computer anhören können. **Mnemonische Symbole in ihrer menschenlesbaren Form geben uns zumindest eine Ahnung, welche Aktion der spätere Maschinenbefehl dann konkret auslöst (siehe oben, „mov“). Der Kreis schließt sich also.**

Zusammenfassung

Hier nochmals der gesamte Weg vom Programmcode einer höheren Programmiersprachen hin zu den Nullen und Einsen (am Beispiel der Programmiersprache C):

```c
int main() {
    int a = 2;
    int b = 3;
    int c = a + b;
    return c;
}
```

zugehöriger C-Code	zugehöriger Assemblercode	Maschinencode (hexadezimal)
`int main() {`	`push rbp` `mov rbp, rsp`	55 48 89 E5
`int a = 2;`	`mov DWORD PTR [rbp-4], 2`	C7 45 FC 02
`int b = 3;`	`mov DWORD PTR [rbp-8], 3`	C7 45 F8 03

`int c = a + b;`	`mov eax, DWORD PTR [rbp-8]`
	`mov edx, DWORD PTR [rbp-4]`
	`add eax, edx`
	`mov DWORD PTR [rbp-12], eax`
		8B 45 F8
		8B 55 FC
		01 D0
		89 45 F4

`return c;`	`mov eax, DWORD PTR [rbp-12]`	8B 45 F4
`}`	`pop rbp` `ret`	5D C3

oder Maschinencode (binär)

01010101
01001000 10001001 11100101

In der Tabelle auf dem nächsten Arbeitsblatt könnt ihr selbst den hexadezimalen Maschinencode in binären Maschinencode mit Nullen und Einsen umwandeln! Ich bin gespannt, was euch bei diesem Arbeitsauftrag auffällt!

Quellen:

https://de.wikipedia.org/wiki/Compiler (17.06.2020)

https://de.wikipedia.org/wiki/Bin%C3%A4rcode (17.06.2020)

http://fbmathe.bbs-bingen.de/Informatik/C_plusplus/grundbegriffe.htm (17.06.2020)

https://de.wikipedia.org/wiki/Opcode (17.06.2020)

http://sparksandflames.com/files/x86InstructionChart.html (17.06.2020)

https://ascii-tabelle.org/ (17.06.2020)

https://de.wikipedia.org/wiki/Maschinensprache (17.06.2020)

http://www.virtualuniversity.ch/software/ascii/ (17.06.2020)

http://mikrocontroller.rahm-home.de/docfiles/Uebungen/2_3_1_Maschinensprache_und_Assemblercode.pdf (17.06.2020)

https://www.kstbb.de/informatik/oo/06/6_1_Programmiersprachen_der_ersten_bis_dritten_Generation.html (17.06.2020)

https://www.it-talents.de/blog/it-talents/der-assembler-befehlssatz (17.06.2020)

https://www.publicdomainpictures.net/de/view-image.php?image=137430&picture=cpu (17.06.2020)